Bash (Unix shell)

Bash is a Unix shell and command language written by Brian Fox for the GNU Project as a free software replacement for the Bourne shell.[8][9] First released in 1989,[10] it has been distributed widely as the default login shell for most Linux distributions and Apple's macOS (formerly OS X). A version is also available for Windows 10.[11] It is also the default user shell in Solaris 11. [12]

Bash is a command processor that typically runs in a text window where the user types commands that cause actions. Bash can also read and execute commands from a file, called a shell script. Like all Unix shells, it supports filename globbing (wildcard matching), piping, here documents, command substitution, variables, and control structures for condition-testing and iteration. The keywords, syntax and other basic features of the language are all copied from sh. Other features, e.g., history, are copied from csh and ksh. Bash is a POSIX-compliant shell, but with a number of extensions.

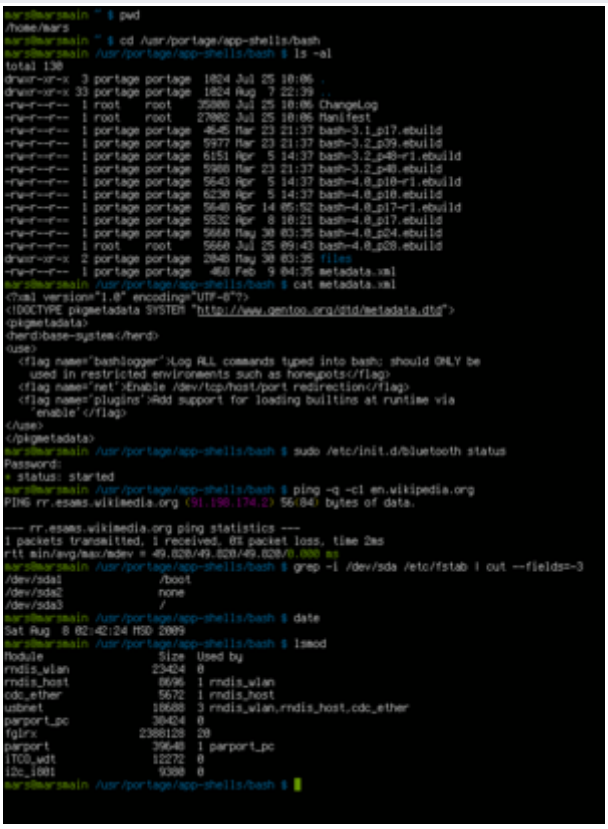
The shell's name is an acronym for *Bourne-again shell*, a pun on the name of the Bourne shell that it replaces[13] and on the common term "born again".[14][15]

A security hole in Bash dating from version 1.03 (August 1989),[16] dubbed Shellshock, was discovered in early September 2014 and quickly led to a range of attacks across the Internet.[17][18][19] Patches to fix the bugs were made available soon after the bugs were identified.

Contents

- History
- Features
  - Brace expansion
  - Startup scripts
    - Execution order of startup files
      - When started as an interactive login shell
      - When a login shell exits
      - When started as an interactive shell (but not a login shell)
    - Comparison with the Bourne shell and csh startup sequences
      - Setting inheritable environment variables
      - Aliases and functions
      - Commands performed only at login and logout
      - Legacy-compatible Bash startup example
    - Operating system issues in Bash startup

Bash



Screenshot of a Bash session

Original author(s)	Brian Fox
Initial release	June 8, 1989
Stable release	5.0 (January 7, 2019) [±][1][2]
Repository	git.savannah.gnu.org/cgit/bash.git
Written in	C
Operating system	Unix-like,[3] macOS (system includes an older GPLv2 version; current version available through third parties)[4] Windows (newer GPLv3+ version)[5][6]
Platform	GNU
Available in	Multilingual (gettext)

Portability
Keyboard shortcuts
Process management
Conditional execution
Bug reporting
See also
References
External links

Type	Unix shell, command language
License	GPLv3+[7]
Website	www.gnu.org/software/bash/

History

Brian Fox began coding Bash on January 10, 1988[20] after Richard Stallman became dissatisfied with the lack of progress being made by a prior developer.[8] Stallman and the Free Software Foundation (FSF) considered a free shell that could run existing shell scripts so strategic to a completely free system built from BSD and GNU code that this was one of the few projects they funded themselves, with Fox undertaking the work as an employee of FSF.[8][21] Fox released Bash as a beta, version .99, on June 8, 1989[10] and remained the primary maintainer until sometime between mid-1992[22] and mid-1994,[23] when he was laid off from FSF[24] and his responsibility was transitioned to another early contributor, Chet Ramey.[25][26][27]

Since then, Bash has become by far the most popular shell among users of Linux, becoming the default interactive shell on that operating system's various distributions (although Almquist shell may be the default scripting shell) and on Apple's macOS.[28][29][30] Bash has also been ported to Microsoft Windows and distributed with Cygwin and MinGW, to DOS by the DJGPP project, to Novell NetWare and to Android via various terminal emulation applications.

In September 2014, Stéphane Chazelas, a Unix/Linux specialist,[31] discovered a security bug in the program. The bug, first disclosed on September 24, was named Shellshock and assigned the numbers CVE-2014-6271, CVE-2014-6277[32] and CVE-2014-7169. The bug was regarded as severe, since CGI scripts using Bash could be vulnerable, enabling arbitrary code execution. The bug was related to how Bash passes function definitions to subshells through environment variables.[33]

Features

The Bash command syntax is a superset of the Bourne shell command syntax. Bash supports brace expansion, command line completion, basic debugging[34] and exception handling (using trap) among other features. Bash can execute the vast majority of Bourne shell scripts without modification, with the exception of Bourne shell scripts stumbling into fringe syntax behavior interpreted differently in Bash or attempting to run a system command matching a newer Bash builtin, etc. Bash command syntax includes ideas drawn from the Korn shell (ksh) and the C shell (csh) such as command line editing, command history, the directory stack, the `$RANDOM` and `$PPID` variables, and POSIX command substitution syntax `$(…)`.

When a user presses the tab key within an interactive command-shell, Bash automatically uses command line completion, since beta version of 2.04[35], to match partly typed program names, filenames and variable names. The Bash command-line completion system is very flexible and customizable, and is often packaged with functions that complete arguments and filenames for specific programs and tasks.

Bash's syntax has many extensions lacking in the Bourne shell. Bash can perform integer calculations ("arithmetic evaluation") without spawning external processes. It uses the `((…))` command and the `$((…))` variable syntax for this purpose. Its syntax simplifies I/O redirection. For example, it can redirect standard output (stdout) and standard error (stderr) at the same time using the `&>` operator. This is simpler to type than the Bourne shell equivalent '`command > file 2>&1`'. Bash supports process substitution using the `<(command)` and `>(command)` syntax, which substitutes the output of (or input to) a command where a filename is normally used. (This is implemented through */proc/fd/* unnamed pipes on systems that support that, or via temporary named pipes where necessary).

When using the 'function' keyword, Bash function declarations are not compatible with Bourne/Korn/POSIX scripts (the Korn shell has the same problem when using 'function'), but Bash accepts the same function declaration syntax as the Bourne and Korn shells, and is POSIX-conformant. Because of these and other differences, Bash shell scripts are rarely runnable under the Bourne or Korn shell interpreters unless deliberately written with that compatibility in mind, which is becoming less common as Linux becomes more widespread. But in POSIX mode, Bash conforms with POSIX more closely.[36]

Bash supports here documents. Since version 2.05b Bash can redirect standard input (stdin) from a "here string" using the <<< operator.

Bash 3.0 supports in-process regular expression matching using a syntax reminiscent of Perl.[37][38]

In February 2009[39] Bash 4.0, introduced support for associative arrays.[36][40] Associative arrays allow a fake support for multi-dimensional (indexed) arrays, in a similar way to AWK. Bash 4.x has not been integrated in newer version of MacOS due to license restrictions.[41]. Associative array example:

```
$ declare -A aa         # declare an associative array 'aa' faking a bi-dimensional indexed array
$ i=1; j=2              # initialize some indices
$ aa[$i,$j]=5           # associate value "5" to key "$i,$j" (i.e. "1,2")
$ echo ${aa[$i,$j]}     # print the stored value at key "$i,$j"
5
```

[42]

Brace expansion

Brace expansion, also called alternation, is a feature copied from the C shell. It generates a set of alternative combinations. Generated results need not exist as files. The results of each expanded string are not sorted and left to right order is preserved:

```
$ echo a{p,c,d,b}e
ape ace ade abe
$ echo {a,b,c}{d,e,f}
ad ae af bd be bf cd ce cf
```

Users should not use brace expansions in portable shell scripts, because the Bourne shell does not produce the same output.

```
$ # A traditional shell does not produce the same output
$ /bin/sh -c 'echo a{p,c,d,b}e'
a{p,c,d,b}e
```

When brace expansion is combined with wildcards, the braces are expanded first, and then the resulting wildcards are substituted normally. Hence, a listing of JPEG and PNG images in the current directory could be obtained using:

```
ls *.{jpg,jpeg,png}     # expands to *.jpg *.jpeg *.png - after which,
                        # the wildcards are processed
echo *.{png,jp{e,}g}    # echo just show the expansions -
                        # and braces in braces are possible.
```

In addition to alternation, brace expansion can be used for sequential ranges between two integers or characters separated by double dots. Newer versions of Bash allow a third integer to specify the increment.

```
$ echo {1..10}
1 2 3 4 5 6 7 8 9 10
$ echo file{1..4}.txt
file1.txt file2.txt file3.txt file4.txt
$ echo {a..e}
a b c d e
$ echo {1..10..3}
1 4 7 10
```

```
$ echo {a..j..3}
a d g j
```

When brace expansion is combined with variable expansion the variable expansion is performed **after** the brace expansion, which in some cases may necessitate the use of the `eval` built-in, thus:

```
$ start=1; end=10
$ echo {$start..$end}  # fails to expand due to the evaluation order
{1..10}
$ eval echo {$start..$end} # variable expansion occurs then resulting string is evaluated
1 2 3 4 5 6 7 8 9 10
```

Startup scripts

When Bash starts, it executes the commands in a variety of dot files. Though similar to Bash shell script commands, which have execute permission enabled and an interpreter directive like `#!/bin/bash`, the initialization files used by Bash require neither

Execution order of startup files

When started as an interactive login shell

Bash reads and executes `/etc/profile` (if it exists). (Often this file calls `/etc/bash.bashrc`)

After reading that file, it looks for `~/.bash_profile`, `~/.bash_login`, and `~/.profile` *in that order*, and reads and executes the first one that exists and is readable.

When a login shell exits

Bash reads and executes `~/.bash_logout` (if it exists).

When started as an interactive shell (but not a login shell)

Bash reads and executes `/etc/bash.bashrc` and then `~/.bashrc` (if it exists). This may be inhibited by using the `--norc` option. The `--rcfile file` option forces Bash to read and execute commands from `file` instead of `~/.bashrc`.

Comparison with the Bourne shell and csh startup sequences

Elements of Bash derive from the Bourne shell and csh. These allow limited startup file sharing with the Bourne shell and provide some startup features familiar to csh users.

Setting inheritable environment variables

The Bourne shell uses the `~/.profile` at login to set environment variables that subprocesses then inherit. Bash can use the `~/.profile` in a compatible way, by executing it explicitly from the Bash-specific `~/.bash_profile` or `~/.bash_login` with the line below. Bash-specific syntax can be kept out of the `~/.profile` to keep the latter compatible with the Bourne shell.

```
. ~/.profile
```

Aliases and functions

These two facilities, aliases from *csh* and the more general functions that largely supersede them from Bourne shell, were not typically inheritable from the login shell, and had to be redefined in each subshell spawned from the login shell. Although there is an `ENV` environment variable that could be applied to the problem, both *ksh* and Bash support per-subshell startup files that address it directly. In Bash, the `~/.bashrc` is called for interactive subshells. If user-defined functions from the `~/.bashrc` are desired in the login shell as well, the `~/.bash_login` can include the line below after any setting up of environment variables:

```
. ~/.bashrc
```

Commands performed only at login and logout

The *csh* supports a `~/.login` file for purposes of tasks performed only during initial login, such as displaying system load, disk status, whether email has come in, logging the login time, etc. The Bourne shell can emulate this in the `~/.profile`, but doesn't predefine a file name. To achieve similar semantics to the csh model, the `~/.bash_profile` can contain the line below, after the environment setup and function setup:

```
. ~/.bash_login
```

Likewise, the *csh* has a `~/.logout` file run only when the login shell exits. The Bash equivalent is `~/.bash_logout`, and requires no special setup. In the Bourne shell, the `trap` built-in can be used to achieve a similar effect.

Legacy-compatible Bash startup example

The skeleton `~/.bash_profile` below is compatible with the Bourne shell and gives semantics similar to csh for the `~/.bashrc` and `~/.bash_login` The `[ -r filename ]` are tests to see if the *filename* exists and is readable, simply skipping the part after the `&&` if it's not.

```
[ -r ~/.profile  ] && . ~/.profile              # set up environment, once, Bourne-sh syntax only
if [ -n "$PS1" ] ; then                         # are we interactive?
   [ -r ~/.bashrc      ] && . ~/.bashrc         # tty/prompt/function setup for interactive shells
   [ -r ~/.bash_login  ] && . ~/.bash_login     # any at-login tasks for login shell only
fi                                              # End of "if" block
```

Operating system issues in Bash startup

Some versions of Unix and Linux contain Bash system startup scripts, generally under the `/etc` directories. Bash calls these as part of its standard initialization, but other startup files can read them in a different order than the documented Bash startup sequence. The default content of the root user's files may also have issues, as well as the skeleton files the system provides to new user accounts upon setup. The startup scripts that launch the X window system may also do surprising things with the user's Bash startup scripts in an attempt to set up user-environment variables before launching the window manager. These issues can often be addressed using a `~/.xsession` or `~/.xprofile` file to read the `~/.profile` — which provides the environment variables that Bash shell windows spawned from the window manager need, such as xterm or Gnome Terminal.

Portability

Invoking Bash with the `--posix` option or stating `set -o posix` in a script causes Bash to conform very closely to the POSIX 1003.2 standard.[43] Bash shell scripts intended for portability should at least take into account the Bourne shell it intends to replace. Bash has certain features that the traditional Bourne shell lacks. They include:[43]

- Certain extended invocation options
- Command substitution using $( ) notation (this feature is part of the POSIX 1003.2 standard though)

- Brace expansion
- Certain array operations, and associative arrays
- The double brackets extended test construct
- The double-parentheses arithmetic-evaluation construct
- Certain string-manipulation operations
- Process substitution
- A Regular Expression matching operator
- Bash-specific builtins
- Coprocesses

A "bashism" is a portion of bash code that does not run properly on other Unix shells.[44]

Keyboard shortcuts

Bash uses readline to provide keyboard shortcuts for command line editing using the default (Emacs) key bindings. Vi-bindings can be enabled by running `set -o vi`[45]

Process management

The Bash shell has two modes of execution for commands: batch, and concurrent mode.

To execute commands in batch (i.e., in sequence) they must be separated by the character ";", or on separate lines:

```
command1; command2
```

in this example, when command1 is finished, command2 is executed.

You can also have a background execution of command1 using (symbol &) at the end of your execution command, and process will be executed in background returning immediately control to your shell and allowing you to keep executing commands.

```
command1 &
```

Or to have a concurrent execution of two command1 and command2, they must be executed in the Bash shell in the following way:

```
command1 & command2
```

In this case command1 is executed in the background & symbol, returning immediately control to the shell that executes command2 in the foreground.

A process can be stopped and returned control to bash by typing Ctrl + z while the process is running in the foreground[46].

A list of all processes, both in the background and stopped, can be achieved by running `jobs`:

```
$ jobs
[1]-  Running                 command1 &
[2]+  Stopped                 command2
```

In the output, the number in brackets refers to the job id. The plus sign signifies the default process for `bg` and `fg`. The text "Running" and "Stopped" refer to the Process state. The last string is the command that started the process.

The state of a process can be changed using various commands. The `fg` command brings a process to the foreground, while the `bg` sets a stopped process running in the background. `bg` and `fg` can take a job id as their first argument, to specify the process to act on. Without one, they use the default process, identified by a plus sign in the output of `jobs`. The `kill` command can be used to end a process prematurely by sending it a signal. The job id must be specified after a percent sign:

```
kill -s SIGKILL %1 or  kill -9 %1
```

Conditional execution

Bash supplies "conditional execution" command separators that make execution of a command contingent on the exit code set by a precedent command. For example:

```
cd "$SOMEWHERE" && ./do_something  || echo "An error occurred"  >&2
```

Where *./do_something* is only executed if the *cd* (change directory) command was "successful" (returned an exit status of zero) and the *echo* command would only be executed if either the *cd* or the *./do_something* command return an "error" (non-zero exit status).

For all commands the exit status is stored in the special variable $?. Bash also supports `if ...;then ...;else ...;fi` and `case $VARIABLE in $pattern)...;;$other_pattern)...;; esac` forms of conditional command evaluation.

Bug reporting

An external command called *bashbug* reports Bash shell bugs. When the command is invoked, it brings up the user's default editor with a form to fill in. The form is mailed to the Bash maintainers (or optionally to other email addresses).[47][48]

See also

- Comparison of command shells

References

1. "Index of /gnu/bash" (https://ftp.gnu.org/gnu/bash/) *ftp.gnu.org*. Retrieved March 19, 2019.
2. Ramey, Chet (January 7, 2019). "Bash-5.0 release available" (https://lists.gnu.org/archive/html/bug-bash/2019-01/msg00063.html). *lists.gnu.org*. Archived (https://web.archive.org/web/20190109012010/http://lists.gnu.org/archive/html/bug-bash/2019-01/msg00063.html) from the original on January 8, 2019 Retrieved March 19, 2019.
3. "Bash FAQ, version 4.14" (https://web.archive.org/web/20180901171316/ftp://ftp.cwru.edu/pub/bash/FAQ). Archived from the original (ftp://ftp.cwru.edu/pub/bash/FAQ) on September 1, 2018 Retrieved April 9, 2016.
4. "Why does Apple ship bash 3.2?" (http://apple.stackexchange.com/questions/208312/why-does-apple-ship-bash-3-2). *apple.stackexchange.com*
5. "Missing source code - GPL compliance? · Issue #107 · Microsoft/WSL" (https://github.com/Microsoft/BashOnWindows/issues/107). *GitHub*.
6. "GNU Bash" (http://www.softpedia.com/get/System/System-Miscellaneous/GNU-Bash.shtml) *Softpedia*. SoftNews. Retrieved April 9, 2016.
7. GNU Project. "README file" (https://www.gnu.org/software/bash/) "Bash is free software, distributed under the terms of the [GNU] General Public License as published by the Free Software Foundation, version 3 of the License (or any later version)."
8. Richard Stallman (forwarded with comments by Chet Ramey) (February 10, 1988). "GNU + BSD = ?" (https://groups.google.com/forum/#!original/comp.unix.questions/iNjWwkyroR8/yedr9yDWSuQJ) Newsgroup: comp.unix.questions (news:comp.unix.questions) Usenet: 2362@mandrill.CWRU.Edu (news:2362@mandrill.CWRU.Edu) Retrieved March 22, 2011. "For a year and a half, the GNU shell was "just about done" The author made repeated promises to deliver what he had done, and never kept them Finally I could no longer believe he would ever deliver anything So Foundation staff member Brian Fox is now implementing an imitation of the Bourne shell".

9. Hamilton, Naomi (May 30, 2008), "The A-Z of Programming Languages: BASH/Bourne-Again Shell" (http://www.computerworld.com.au/article/222764/a-z_programming_languages_bash_bourne-again_shell/?pp=2&fp=16&fpid=1), *Computerworld*. 2, retrieved March 21, 2011, "When Richard Stallman decided to create a full replacement for the then-encumbered Unix systems, he knew that he would eventually have to have replacements for all of the common utilities, especially the standard shell, and those replacements would have to have acceptable licensing."
10. Brian Fox (forwarded by Leonard H. Tower Jr.) (June 8, 1989). "Bash is in beta release!" (https://groups.google.com/group/gnu.announce/msg/a509f48fb298c35?hl=en). Newsgroup: gnu.announce (news:gnu.announce). Retrieved October 28, 2010.
11. "How to install Bash shell command-line tool on Windows 10" (http://www.windowscentral.com/how-install-bash-shell-command-line-windows-10) September 28, 2016.
12. "User Environment Feature Changes" (https://docs.oracle.com/cd/E23824_01/html/E24456/userenv-1.html) Oracle.
13. C Programming (http://www.ddj.com/cpp/184404693) by Al Stevens, Dr. Dobb's Journal, July 1, 2001
14. Richard Stallman (November 12, 2010). "About the GNU Project" (https://www.gnu.org/gnu/thegnuproject.html). Free Software Foundation. Archived (https://web.archive.org/web/20110424064815/https://www.gnu.org/gnu/thegnuproject.html) from the original on April 24, 2011. Retrieved March 13, 2011. ""Bourne Again Shell" is a play on the name *Bourne Shell*, which was the usual shell on Unix."
15. Gattol, Markus (March 13, 2011), *Bourne-again Shell* (https://web.archive.org/web/20110309092607/http://www.markus-gattol.name/ws/bash.html) archived from the original (http://www.markus-gattol.name/ws/bash.html) on March 9, 2011, retrieved March 13, 2011, "The name is a pun on the name of the Bourne shell (sh), an early and important Unix shell written by Stephen Bourne and distributed with Version 7 Unix circa 1978, and the concept of being "born again"."
16. Chazelas, Stephane (October 4, 2014). "oss-sec mailing list archives" (http://seclists.org/oss-sec/2014/q4/102) *Seclists.org*. Retrieved October 4, 2014.
17. Leyden, John (September 24, 2014). "Patch Bash NOW: 'Shell Shock' bug blasts OS X, Linux systems wide open" (https://www.theregister.co.uk/2014/09/24/bash_shell_vuln/) *The Register*. Retrieved September 25, 2014.
18. Perlroth, Nicole (September 25, 2014). "Security Experts Expect 'Shellshock' Software Bug in Bash to Be Significant" (https://www.nytimes.com/2014/09/26/technology/security-experts-expect-shellshock-software-bug-to-be-significant.html). *The New York Times*. Retrieved September 25, 2014.
19. Seltzer, Larry (September 29, 2014). "Shellshock makes Heartbleed look insignificant" (https://web.archive.org/web/20160514191755/http://www.zdnet.com/article/hackers-jump-on-the-shellshock-bash-bandwagon/) *ZDNet*. Archived from the original (http://www.zdnet.com/shellshock-makes-heartbleed-look-insignificant-7000034143/) on May 14, 2016.
20. Brian Fox (August 29, 1996), *shell.c* (http://ftp.gnu.org/gnu/bash/bash-1.14.7.tar.gz), Free Software Foundation "Birthdate: Sunday, January 10th, 1988. Initial author: Brian Fox"
21. Richard Stallman (October 3, 2010). "About the GNU Project" (https://www.gnu.org/gnu/thegnuproject.html). Free Software Foundation. Archived (https://web.archive.org/web/20110424064815/https://www.gnu.org/gnu/thegnuproject.html) from the original on April 24, 2011. Retrieved March 21, 2011. "Free Software Foundation employees have written and maintained a number of GNU software packages. Two notable ones are the C library and the shell. ... We funded development of these programs because the GNU Project was not just about tools or a development environment. Our goal was a complete operating system, and these programs were needed for that goal."
22. len (g...@prep.ai.mit.edu) (April 20, 1993). "January 1993 GNU's Bulletin" (https://groups.google.com/group/gnu.misc.discuss/msg/4f42c739cd7e8bd8). Newsgroup: gnu.announce (news:gnu.announce). Usenet: gnusenet930421bulletin@prep.ai.mit.edu (news:gnusenet930421bulletin@prep.ai.mit.edu). Retrieved October 28, 2010.
23. Ramey, Chet (August 1, 1994). "Bash - the GNU shell (Reflections and Lessons Learned)" (http://www.linuxjournal.com/article/2800#N0xa50890.0xb46380). *Linux Journal*. Archived (https://web.archive.org/web/20081205082152/http://www.linuxjournal.com/article/2800) from the original on December 5, 2008. Retrieved November 13, 2008.
24. Chet Ramey (October 31, 2010), *Dates in your Computerworld interview* (https://www.scribd.com/doc/40556434/2010-10-31-Chet-Ramey-Early-Bash-Dates), retrieved October 31, 2010
25. Chet Ramey (June 12, 1989). "Bash 0.99 fixes & improvements" (https://groups.google.com/group/gnu.bash.bug/msg/1fc7b688f5d44438?hl=en). Newsgroup: gnu.bash.bug (news:gnu.bash.bug). Retrieved November 1, 2010.
26. Chet Ramey (July 24, 1989). "Some bash-1.02 fixes" (https://groups.google.com/group/gnu.bash.bug/msg/072a03645663caea?hl=en). Newsgroup: gnu.bash.bug (news:gnu.bash.bug). Retrieved October 30, 2010.

27. Brian Fox (March 2, 1990). "Availability of bash 1.05" (https://groups.google.com/group/gnu.bash.bug/msg/e6112ccc8866e2f4?hl=en). Newsgroup: gnu.bash.bug (news:gnu.bash.bug). Retrieved October 30, 2010.
28. Foster-Johnson, Eric; Welch, John C.; Anderson, Micah (April 2005). *Beginning Shell Scripting* (https://books.google.com/books?id=dwIRERUpQPEC&pg=PA6&dq=bash+most+popular+unix+shell&hl=en&sa=X&ved=0ahUKEwi_3fnR2pPNAhUX3GMKHTEdAZoQ6AEIQTAF#v=onepage&q=bash%20most%20popular%20unix%20shell&f=false). John Wiley & Sons, Inc. p. 6. ISBN 978-0-7645-9791-6. Retrieved June 6, 2016. "Bash is by far the most popular shell and forms the default shell on Linux and Mac OSX systems".
29. Danesh, Arman; Jang, Michael (February 2006). *Mastering Linux* (https://books.google.com/books?id=tIjrVYbZmUAC&pg=PA363&dq=bash+most+popular+unix+shell&hl=en&sa=X&ved=0ahUKEwjnipq_3JPNAhUH52MKHUD-DA04ChDoAQgpMAA#v=onepage&q=bash%20most%20popular%20unix%20shell&f=false). John Wiley & Sons, Inc. p. 363. ISBN 978-0-7821-5277-7. Retrieved June 6, 2016. "The Bourne Again Shell (bash) is the most common shell installed with Linux distributions".
30. Bresnahan, Christine; Blum, Richard (April 2015). *CompTIA Linux+ Powered by Linux Professional Institute Study Guide: Exam LX0-103 and Exam LX0-104* (https://books.google.com/books?id=2P3zBgAAQBAJ&pg=PA5&dq=%22most+popular%22+linux+shell&hl=en&sa=X&ved=0ahUKEwiDhpnX4JPNAhWCKGMKHQCmCoEQ6AEIRjAH#v=onepage&q=%22most%20popular%22%20linux%20shell&f=false) (3rd ed.). John Wiley & Sons, Inc. p. 5. ISBN 978-1-119-02122-3. Retrieved June 6, 2016. "In Linux, most users run bash because it is the most popular shell".
31. https://www.linkedin.com/pub/st%C3%A9phane-chazelas/7/2a2/834
32. "CVE-2014-6277" (https://cve.mitre.org/cgi-bin/cvename.cgi?name=2014-6277). *cve.mitre.org*.
33. Huzaifa Sidhpurwala (September 24, 2014). "Bash specially-crafted environment variables code injection attack" (https://securityblog.redhat.com/2014/09/24/bash-specially-crafted-environment-variables-code-injection-attack/). Red Hat.
34. http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_02_03.html
35. http://www.caliban.org/bash/index.shtml
36. "6.11 Bash POSIX Mode" (https://www.gnu.org/software/bash/manual/html_node/Bash-POSIX-Mode.html). *The GNU Bash Reference Manual, for Bash, Version 4.1* (https://www.gnu.org/software/bash/manual/html_node/index.html), December 23, 2009, archived (https://web.archive.org/web/20101203065400/https://www.gnu.org/software/bash/manual/html_node/index.html) from the original on December 3, 2010, retrieved October 26, 2010
37. "Advanced Bash-Scripting Guide" (http://www.tldp.org/LDP/abs/html/bashver3.html#BASH3REF). *www.tldp.org*. Section 37.2 (Bash, version 3) Retrieved March 5, 2017.
38. The syntax matches that shown on the `regex(7) (http://www.tin.org/bin/man.cgi?section=7&topic=regex)` man page.
39. http://tldp.org/LDP/abs/html/bashver4.html
40. "The shell provides associative array variables, with the appropriate support to create, delete, assign values to, and expand them." http://tiswww.case.edu/php/chet/bash/NEWS
41. https://apple.stackexchange.com/questions/193411/update-bash-to-version-4-0-on-osx
42. https://www.gnu.org/software/bash/manual/html_node/Arrays.html
43. Mendel Cooper. "Portability Issues" (http://tldp.org/LDP/abs/html/portabilityissues.html). *The Linux Documentation Project*. ibiblio.org.
44. "checkbashisms(1) - Linux man page" (https://linux.die.net/man/1/checkbashisms). *linux.die.net*.
45. "BASH Help - A Bash Tutorial" (http://www.hypexr.org/bash_tutorial.php#emacs) Hypexr.org. October 5, 2012 Retrieved July 21, 2013.
46. https://www.gnu.org/software/bash/manual/bash.html#index-background
47. bashbug(1) (http://linux.die.net/man/1/bashbug) die.net
48. "Linux / Unix Command: bashbug" (https://developer.apple.com/library/prerelease/mac/documentation/Darwin/Reference/ManPages/man1/bashbug.1.html) apple.com

External links

- Official website
- 2008 interview with GNU Bash's maintainer, Chet Ramey
- MPI-Bash: A MPI-enabled plugin for the Bourne-Again Shell by Scott Pakin

Retrieved from 'https://en.wikipedia.org/w/index.php?title=Bash_(Unix_shell)&oldid=893487687'

This page was last edited on 21 April 2019, at 18:23 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.